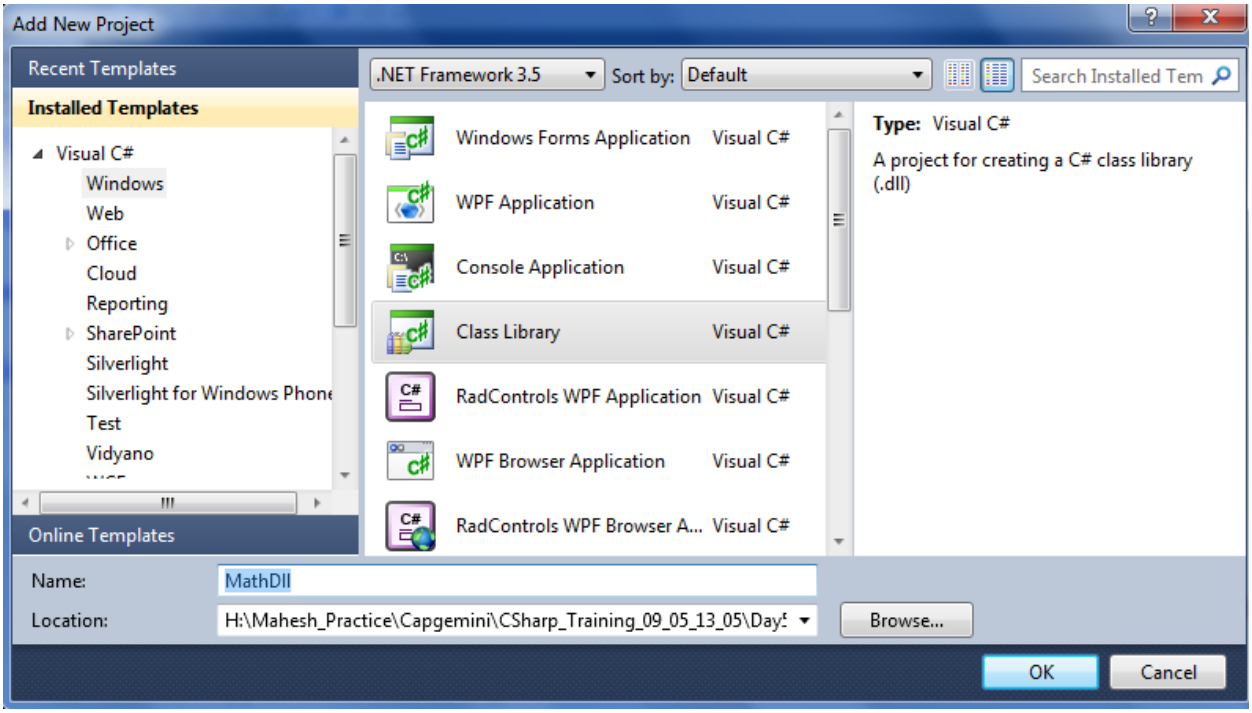
Working with Reflection

In this lab we will see how to work with reflection.

Step 1: Open VS20-8 and create new blank solution. Name it as 'Reflection_Sample'. In this project add a new Class library project, name it as 'MathDll' as below:



Step 2: Rename Class1.cs to MathClass.cs and write the below code in it:

```
namespace MathDll
{
    public class MathClass
    {
        public int Add(int x, int y)
        {
            return x + y;
        }
        public int Sub(int x, int y)
        {
            return x - y;
        }

        public int Mul(int x, int y)
        {
            return x * y;
        }
    }

    public class MathClassNew
    {
```

```
        public int Add(int x, int y)
        {
            return x + y;
        }

        public int Sub(int x, int y)
        {
            return x - y;
        }
    }
}
```

The above code define two classes with some mathematical methods.

Step 3: Build the application.

Step 4: In the same solution created in Step 1, add a new console application, name it as 'CS_Reflection'.

Step 5: In the Program.cs write the below code:

```
using System;

using System.Reflection;

namespace CS_Reflection
{
    class Program
    {
        static void Main(string[] args)
        {

            //Load the Dll from the Path
            //Below please write your MathDll.dll e.g. If it is available on
            // c:\ then the path following method will be
            //Assembly.LoadFrom(@"C:\MathDll.dll")
            Assembly asm = Assembly.LoadFrom(@"<Write your Path for
MathDll.dll>");

            //Read all Types from Assembly

            Type[] arrTypes = asm.GetTypes();

            //Declare Array for Method and Parameters

            MethodInfo[] arrMethods;
            ParameterInfo[] arrParameters;

            foreach (Type t in arrTypes)
            {
                Console.WriteLine("The Current Type is :" + t.FullName);

                //Create the instance
                object dynamicObject = asm.CreateInstance(t.FullName);

                //Read all public , Instance and DeclaredOnly Methods
                arrMethods =
t.GetMethods(BindingFlags.Public|BindingFlags.Instance|BindingFlags.DeclaredO
```

```
nly);

                //Loop for Method
                foreach (MethodInfo m in arrMethods)
                {
                    Console.WriteLine(m.Name);

                    arrParameters = m.GetParameters();
                    Console.WriteLine();
                    //Iterate through All Parameters
                    foreach (ParameterInfo par in arrParameters)
                    {
                        Console.WriteLine("Name :" + par.Name + "Data Type :"
+ par.ParameterType + " Positio :" + par.Position);
                    }
                    //Invoke Method Public, Instance, Declared in the class
                    //Pass parameters as object array
                    object Result = t.InvokeMember(m.Name,
BindingFlags.DeclaredOnly | BindingFlags.Public | BindingFlags.Instance |
BindingFlags.InvokeMethod, null, dynamicObject, new object[] { 5, 4 });
                    Console.WriteLine(m.Name + "=" + Result);
                }

            }
            Console.ReadLine();
        }
    }
}
```

If you see the above code, it uses System.Reflection namespace. Note all "**GREEN**" comments.

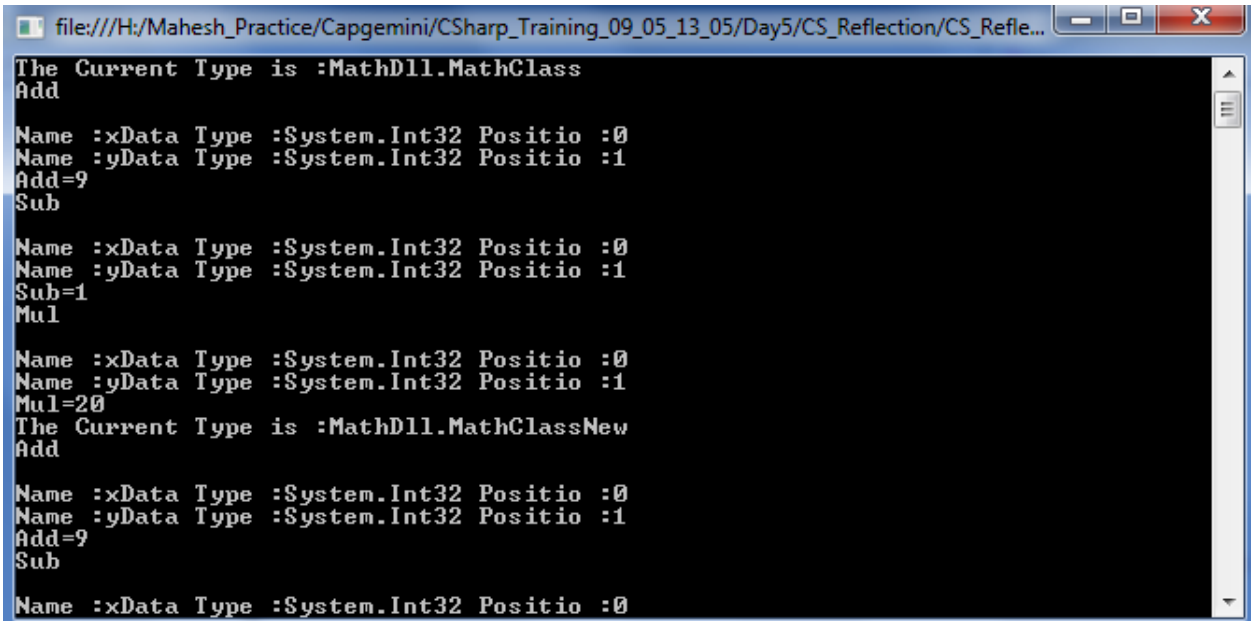
The above code uses following classes:

- Assembly: This class used to Load the assembly.
- Type: Class to read all types (classes, methods etc) from the loaded assembly.
- MethodInfo: Class used to read methods information from the class read from the assembly.
- ParameterInfo: Class used to work with parameters passed to methods.

Also the above code uses:

- BindingFlags: This Enumaeration used to define the search criteria for methods to search from the class e.g.Public, Instance etc.
- CreateInstance() method of the Assembly class is used to create an object of the class using its FullName (Fully qualify name <NamespaceName.className>).
- InvokeMember () method of the Type class is used to invoke the method for the execution.

Step 6: Run the application, the below result will be shown:


file:///H:/Mahesh_Practice/Capgemini/CSharp_Training_09_05_13_05/Day5/CS_Reflection/CS_Refle...
The Current Type is :MathDll.MathClass
Add
Name :xData Type :System.Int32 Positio :0
Name :yData Type :System.Int32 Positio :1
Add=9
Sub
Name :xData Type :System.Int32 Positio :0
Name :yData Type :System.Int32 Positio :1
Sub=1
Mul
Name :xData Type :System.Int32 Positio :0
Name :yData Type :System.Int32 Positio :1
Mul=20
The Current Type is :MathDll.MathClassNew
Add
Name :xData Type :System.Int32 Positio :0
Name :yData Type :System.Int32 Positio :1
Add=9
Sub
Name :xData Type :System.Int32 Positio :0